ARTIFICIAL INTELLIGENCE

(CSC 462)

LAB # 11

NAME: MUAAZ BIN MUKHTAR

REG NO: FA21-BSE-045

CLASS & SECTION: BSSE-5A

SUBMITTED TO: SIR WAQAS ALI

DATE SUBMITTED: 27-11-2023

Department of Computer Science

Lab Task :

Imagine an 8 queen problem, where the goal is to place 8 queens on an 8 X 8 board such that no two queens are on the same row or column or diagonal. (Before proceeding, kindly refer to lectures). A sample state is shown below.

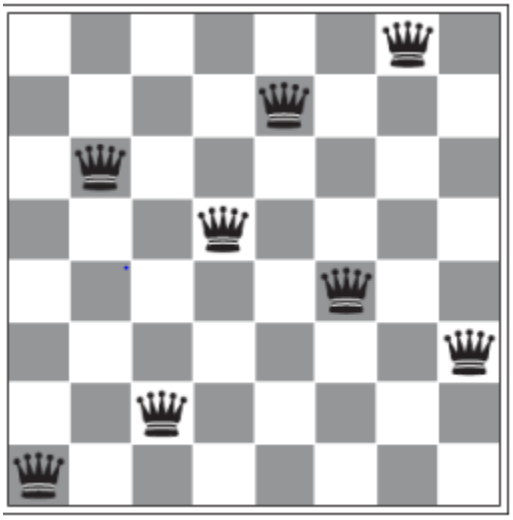

Figure 34 - 8 Queen Problem

Code:

```python
import random

def generate_population(population_size):
    population = []
    for _ in range(population_size):
        chromosome = random.sample(range(8), 8)
        population.append(chromosome)
    return population

def fitness(chromosome):
    clashes = 0
    for i in range(len(chromosome)):
        for j in range(i + 1, len(chromosome)):
            if chromosome[i] == chromosome[j] or abs(i - j) == abs(chromosome[i] - chromosome[j]):
                clashes += 1
    return clashes

def crossover(parent1, parent2):
    crossover_point = random.randint(1, len(parent1) - 1)
    child1 = parent1[:crossover_point] + [gene for gene in parent2 if gene not in parent1[:crossover_point]]
    child2 = parent2[:crossover_point] + [gene for gene in parent1 if gene not in parent2[:crossover_point]]
    return child1, child2
```

```
def mutate(chromosome, mutation_rate):
    if random.random() < mutation_rate:
        swap_positions = random.sample(range(8), 2)
        chromosome[swap_positions[0]], chromosome[swap_positions[1]] = chromosome[swap_positions[1]], chromosome[swap_positions[0]]
    return chromosome

def genetic_algorithm(population_size, generations, mutation_rate):
    population = generate_population(population_size)

    for generation in range(generations):
        population = sorted(population, key=lambda x: fitness(x))
        if fitness(population[0]) == 0:
            print("Solution found in generation", generation)
            return population[0]

        new_population = []
```

```
        for _ in range(population_size // 2):
            parent1 = random.choice(population[:population_size // 2])
            parent2 = random.choice(population[:population_size // 2])
            child1, child2 = crossover(parent1, parent2)
            child1 = mutate(child1, mutation_rate)
            child2 = mutate(child2, mutation_rate)
            new_population.extend([child1, child2])

        population = new_population

    print("No solution found.")
    return None
```

```
if __name__ == "__main__":
    population_size = 100
    generations = 1000
    mutation_rate = 0.1

    solution = genetic_algorithm(population_size, generations, mutation_rate)
    if solution:
        print("Solution:", solution)
```

Output:

```
('Solution found in generation', 5)
('Solution:', [2, 5, 3, 1, 7, 4, 6, 0])

Process finished with exit code 0
```

.